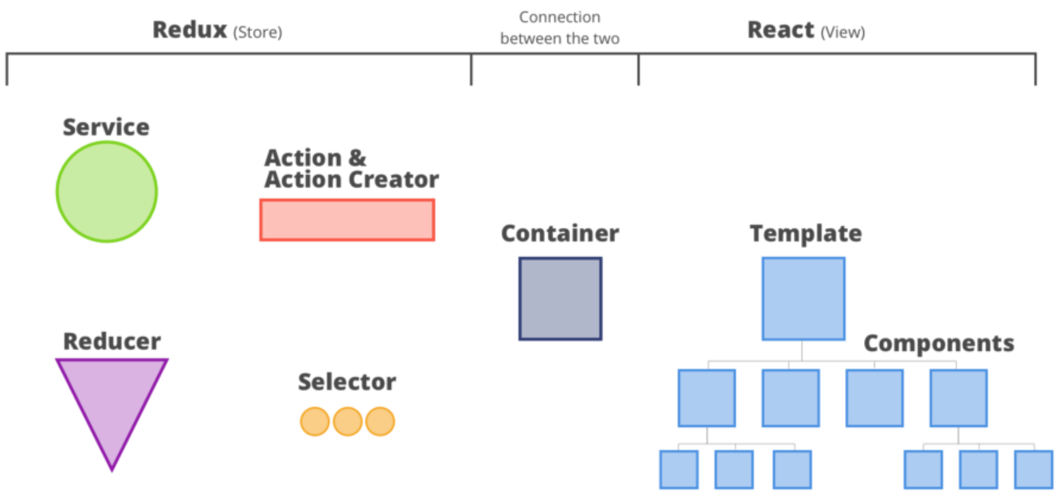
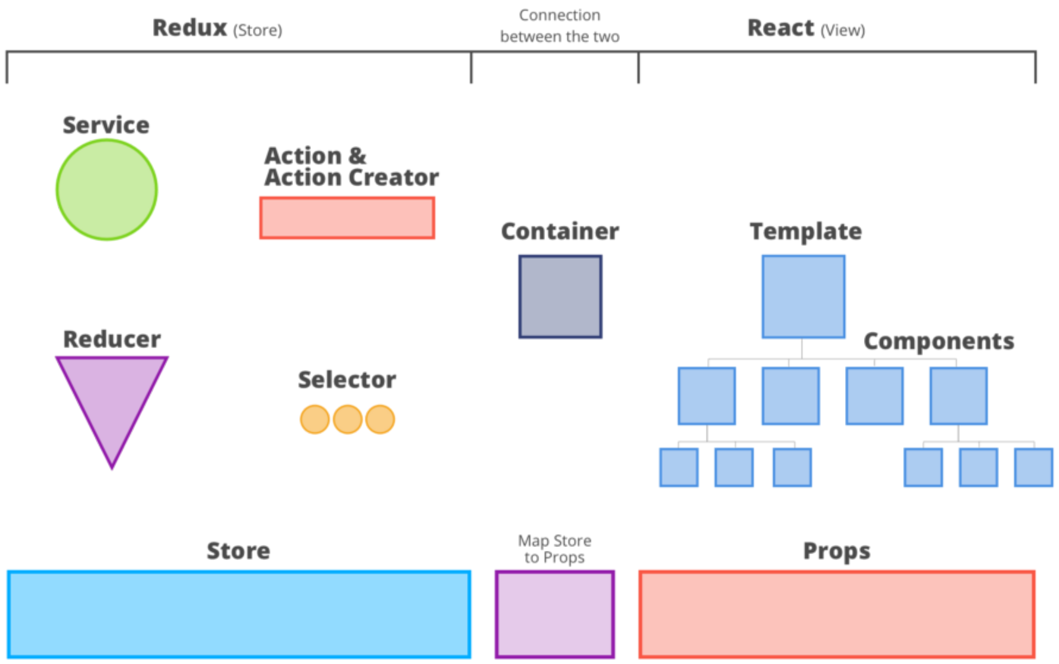
Date=21/12/2020
Lecture By= Akash Handa
Subject ⇒ Redux

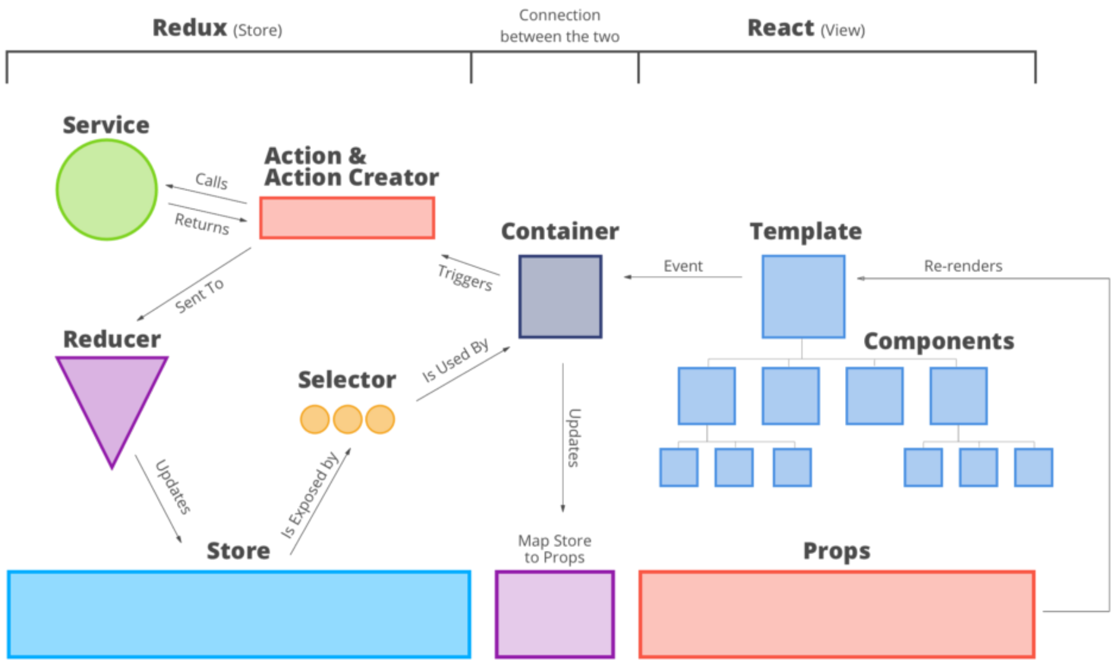
IN PREVIOUS LECTURE (QUICK RECAP) Date-18/12/2020	**In Today's Lecture (Overview)**
Higher-Order Components	What Is Redux?? And Why?? Actions & Action Creators Reducer Selector

What Is Redux?? And Why??

Redux is used mostly for application state management. To summarize it, Redux maintains the state of an entire application in a single immutable state tree (object), which can't be changed directly. When something changes, a new object is created (using actions and reducers).




Redux (Store)
Connection between the two
React (View)
Service
Action & Action Creator
Container
Template
Components
Reducer
Selector
Store
Map Store to Props
Props


Redux (Store)
Connection between the two
React (View)
Service
Calls
Returns
Action & Action Creator
Container
Template
Event
Re-renders
Triggers
Sent To
Components
Reducer
Is Used By
Selector
Updates
Updates
Is Exposed by
Store
Map Store to Props
Props

Actions & Action Creators

I put these together because they are kinda referred to as each other. Often when people are referring to actions they really mean action creators. What is the difference?

Action: This is an object that contains the type of action and the state that was changed because of the action.

Action Creator: This is the code that is called to create an action and send it along to the reducer.

I kinda think of the Action as an event for redux. When an event is fired there is an event type (like onClick, onHover, etc.) and has an event object that contains the data from the event. Well an action is kinda the same, it has a type and the data.

Let's look at how Containers and Actions play together. This is the container file:

```
//Step 1: Pull in the function for the action.
import { getArticles } from '../store/articleList/actions.js';
//Step 2: This connects the action, getArticles, to dispatch the action.
const mapActionsToDispatch = (dispatch) => ({
    loadArticles: () => dispatch( getArticles() )
});
//Step 3: This takes the connected action and merges it to a prop for
React.
const mergeProps = (state, actions) => ({
    ...state,
    ...actions,
    loadArticles: () => actions.loadArticles()
});
//Step 4: Merging it to a prop makes it available in the component,
// in this case on the componentDidMount life cycle event.
// This could just as easily be a click, submit, type, or other type of
event.
```

```
class LandingContainer extends ReactComponent {
    componentDidMount() {
        const {loadArticles} = this.props;
        loadArticles();
    }
    render() {
        return <Landing {...this.props} />
    }
}
```

If you follow the structure you can see how an action get mapped to a property in a component. The file above was rearranged a little so you could see the flow more easily.

Now lets look at the action:

```
// This is the connection to the Reducer. This is where the action type is connected.
import {ADD_ARTICLE, CLEAR_ARTICLES} from './reducers.js';
// This connects redux to the API.
import getArticleList from './services.js';
/* Sync Action Creators */
//By default all actions are synchronous. But we need to call an outside API.
// Here is where the action is created, a type, with a set of data.
export const addArticle = (article = []) => ({
    type: ADD_ARTICLE,
    article: article
});
// This is also a creator, a simple one. All it will do is remove what ever state is in the store for articles.
export const clearArticles = () => ({type: CLEAR_ARTICLES});
/* Async Action Creators */
// Here is the async creator. This is using Thunk that allows us to pass a function as the data in an action.
// it gets run on the other side. Here we calling our service, getArticleList, that returns a promise.
// then we dispatch the action when the promise is resolved.
export const getArticles = () => (dispatch) => {
    getArticleList()
    .then( (articles) => {
        articles.forEach((article) => {
            dispatch(addArticle(article))
        })
    })
};
```

That is how the action is connected to a service, connected to the container, and gets data. But how it sends the data on… for that we need to move to the reducer.

Reducer

The key thing to know about a reducer is that for every dispatch every reducer is called and given the dispatched action. Then it is up to the reducer to handle it or pass it on.

```
// This is declaring the types of actions.
export const ADD_ARTICLE = `ADD_ARTICLES`;
export const CLEAR_ARTICLES = `CLEAR_ARTICLES`;
// This will be called for every dispatch. It is passed the type of
action.
export default function(state = [], {type, article } ) {
    // Here I ignore or handle each action type.
    switch(type) {
        case CLEAR_ARTICLES:
            return [];

        case ADD_ARTICLE:
            return [
                ...state,
                article
            ];
        // if I don't care about the action I just pass along the state
that was given.
        default:
            return state;
    }
}
```

The other part of the reducer that is really important to know is that you don't manipulate state. You create new state. So rather than doing something like this `state.property = "some new value"` instead you would do `return [...state, property]`. State then goes into your store.

Selector

The selector is how you would get data out of your store in the container. As I show the code it may not seem very valuable.

```
//Selector.js
export default (state) => (state.property)
//Container.js
import selectArticles from '../store/articleList/selectors.js';
const mapStateToProps = (state) => {
    return {
        // React is looking for the tiles property in the template, here
we map articles to tiles.
        tiles: selectArticles(state)
    }
}
```

Here is showing how the selector is defined and used in the container. But why use `selectArticles(state)` here instead of `state.property` directly? Because the container shouldn't care about how the store is organized. I can change and do minor tweaks in the selector before it is sent on to the container. It has a lot of power.